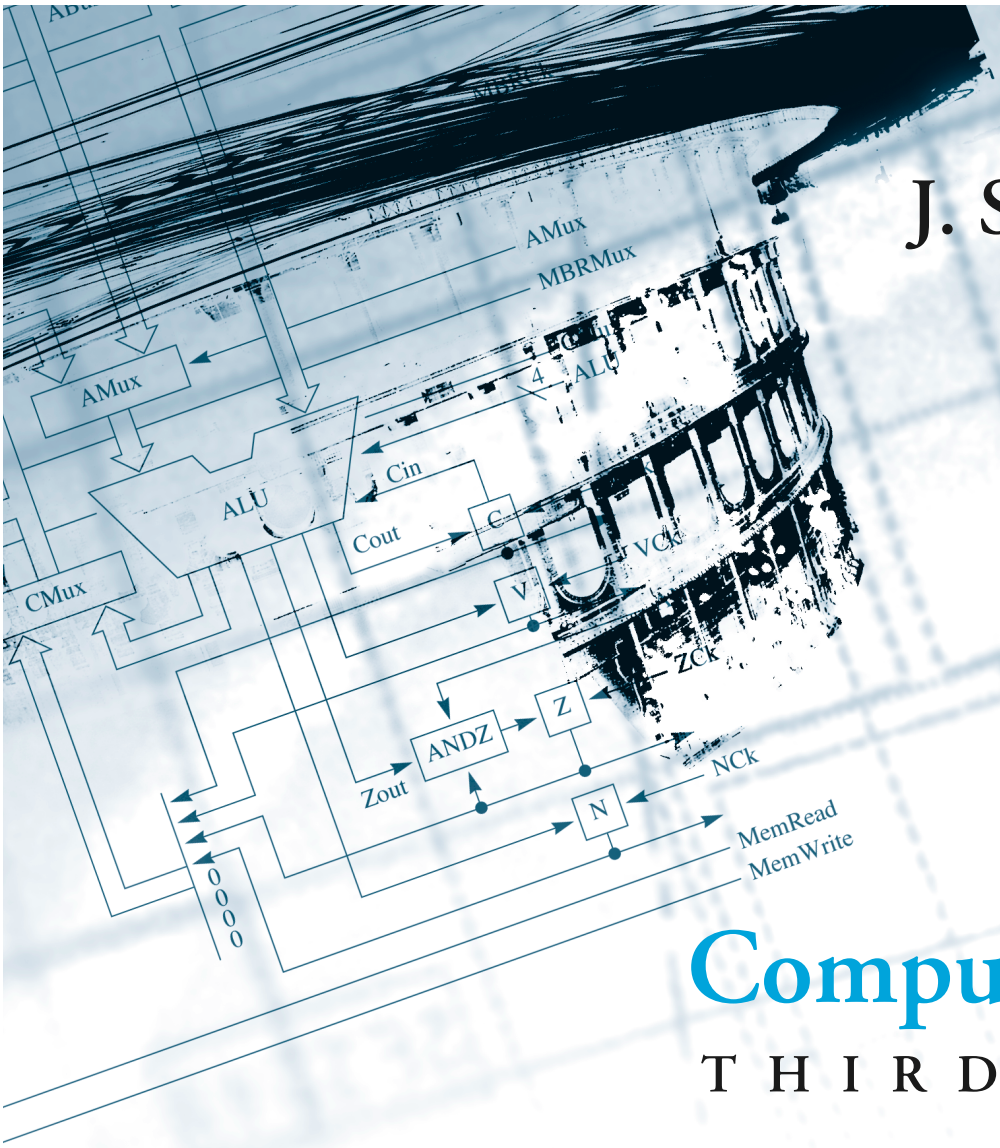
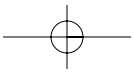


J. Stanley Warford
Pepperdine University

Computer Systems

THIRD EDITION

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE



World Headquarters
Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
www.jbpub.com

Jones and Bartlett Publishers
Canada
2406 Nikanna Road
Mississauga, ON L5C 2W6
CANADA

Jones and Bartlett Publishers
International
Barb House, Barb Mews
London W6 7PA
UK

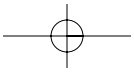
Library of Congress Cataloging-in-Publication Data

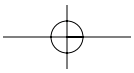
Warford, J. Stanley, 1944-
Computer systems / Stanley Warford.— 3rd ed.
p. cm.
ISBN 0-7637-3239-7 (alk. paper)
1. Computer systems. I. Title.

QA76.W2372 2004
004—dc22 2004019599

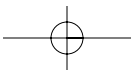
Acquisitions Editor: Timothy Anderson
Production Director: Amy Rose
Marketing Manager: Matthew Payne
Editorial Assistant: Lesley Chiller
Manufacturing Buyer: Therese Bräuer
Cover Design: Kristin E. Ohlin
Composition: Auburn Associates, Inc.
Technical Artist: Smolinski Studios
Printing and Binding: Edwards Brothers
Cover Printing: Edwards Brothers

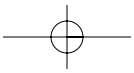
Printed in the United States of America
09 08 07 06 05 10 9 8 7 6 5 4 3 2 1

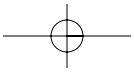




This book is dedicated to my mother,
Susan Warford.







Preface

Computer Systems offers a clear, detailed, step-by-step exposition of the central ideas in computer organization, assembly language, and computer architecture. The book is based in large part on a virtual computer, Pep/8, which is designed to teach the basic concepts of the classic von Neumann machine. The strength of this approach is that the central concepts of computer science are taught without getting entangled in the many irrelevant details that often accompany such courses. This approach also provides a foundation that encourages students to think about the underlying themes of computer science. Breadth is achieved by emphasizing computer science topics that are related to, but not usually included in, the treatment of hardware and its associated software.

Summary of Contents

Computers operate at several levels of abstraction; programming at a high level of abstraction is only part of the story. This book presents a unified concept of computer systems based on the level structure of Figure P.1.

The book is divided into seven parts corresponding to the seven levels of Figure P.1:

Level App7	Applications
Level HOL6	High-order languages
Level ISA3	Instruction set architecture
Level Asmb5	Assembly
Level OS4	Operating system
Level LG1	Logic gate
Level Mc2	Microcode

The text generally presents the levels top-down, from the highest to the lowest. Level ISA3 is discussed before Level Asmb5 and Level LG1 is discussed before

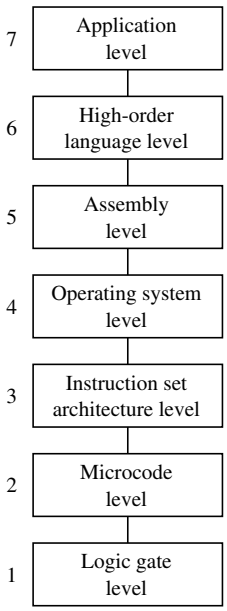
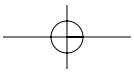


Figure **P.1**
The level structure of a typical computer system.



Level Mc2 for pedagogical reasons. In these two instances, it is more natural to revert temporarily to a bottom-up approach so that the building blocks of the lower level will be in hand for construction of the higher level.

Level App7 Level App7 is a single chapter on application programs. It presents the idea of levels of abstraction and establishes the framework for the remainder of the book. A few concepts of relational databases are presented as an example of a typical computer application. It is assumed that students have experience with text editors or word processors.

Level HOL6 Level HOL6 consists of one chapter, which reviews the C++ programming language. The chapter assumes that the student has experience in some imperative language, such as Java or C, not necessarily C++. Advanced features of C++, including object-oriented concepts, are avoided. The instructor can readily translate the C++ examples to other common Level HOL6 languages if necessary.

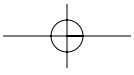
The topic of recursion is treated in this chapter because it depends on the mechanism of memory allocation on the run-time stack. A fairly detailed explanation is given on the details of the memory allocation process for function calls, because this mechanism is revisited at a lower level of abstraction later in the book.

Level ISA3 Level ISA3 is the instruction set architecture level. Its two chapters describe Pep/8, a virtual computer designed to illustrate computer concepts. The Pep/8 computer is a classical von Neumann machine. The CPU contains an accumulator, an index register, a program counter, a stack pointer, and an instruction register. It has eight addressing modes: immediate, direct, indirect, stack-relative, stack-relative deferred, indexed, stack-indexed, and stack-indexed deferred. The Pep/8 operating system, in simulated read-only memory (ROM), can load and execute programs in hexadecimal format from students' text files. Students run short programs on the Pep/8 simulator and learn that executing a store instruction to ROM does not change the memory value.

Students learn the fundamentals of information representation and computer organization at the bit level. Because a central theme of this book is the relationship of the levels to one another, the Pep/8 chapters show the relationship between the ASCII representation (Level ISA3) and C++ variables of type `char` (Level HOL6). They also show the relationship between two's complement representation (Level ISA3) and C++ variables of type `int` (Level HOL6).

Level Asmb5 Level Asmb5 is the assembly level. The text presents the concept of the assembler as a translator between two levels—assembly and machine. It introduces Level Asmb5 symbols and the symbol table.

The unified approach really comes into play here. Chapters 5 and 6 present the compiler as a translator from a high-order language to assembly language. Previously, students learned a specific Level HOL6 language, C++, and a specific von Neumann machine, Pep/8. These chapters continue the theme of relationships between the levels by showing the correspondence between (a) assignment state-



ments at Level HOL6 and load/store instructions at Level Asmb5, (b) loops and `if` statements at Level HOL6 and branching instructions at Level Asmb5, (c) arrays at Level HOL6 and indexed addressing at Level Asmb5, (d) procedure calls at Level HOL6 and the run-time stack at Level Asmb5, (e) function and procedure parameters at Level HOL6 and stack-relative addressing at Level Asmb5, (f) `switch` statements at Level HOL6 and jump tables at Level Asmb5, and (g) pointers at Level HOL6 and addresses at level Asmb5.

The beauty of the unified approach is that the text can implement the examples from the C++ chapter at this lower level. For example, the run-time stack illustrated in the recursive examples of Chapter 2 corresponds directly to the hardware stack in Pep/8 main memory. Students gain an understanding of the compilation process by translating manually between the two levels.

This approach provides a natural setting for the discussion of central issues in computer science. For example, the book presents structured programming at Level HOL6 versus the possibility of unstructured programming at Level Asmb5. It discusses the goto controversy and the structured programming/efficiency tradeoff, giving concrete examples from languages at the two levels.

Chapter 7, Language Translation Principles, introduces students to computer science theory. Now that students know intuitively how to translate from a high-level language to assembly language, we pose the fundamental question underlying all of computing: What can be automated? The theory naturally fits in here because students now know what a compiler (an automated translator) must do. They learn about parsing and finite state machines—deterministic and nondeterministic—in the context of recognizing C++ and Pep/8 assembly language tokens. This chapter includes an automatic translator between two small languages, which illustrates lexical analysis, parsing, and code generation. The lexical analyzer is an implementation of a finite state machine. What could be a more natural setting for the theory?

Level OS4 Level OS4 consists of two chapters on operating systems. Chapter 8 is a description of process management. Two sections, one on loaders and another on trap handlers, illustrate the concepts with the Pep/8 operating system. Five instructions have unimplemented opcodes that generate software traps. The operating system stores the process control block of the user's running process on the system stack, and the interrupt service routine interprets the instruction. The classic state transition diagram for running and waiting processes in an operating system is thus reinforced with a specific implementation of a suspended process. The chapter concludes with a description of concurrent processes and deadlocks. Chapter 9 describes storage management, both main memory and disk memory.

Level LG1 Level LG1 uses two chapters to present combinational and sequential circuits. Chapter 10 emphasizes the importance of the mathematical foundation of computer science by starting with the axioms of boolean algebra. It shows the relationship between boolean algebra and logic gates, and then describes some common SSI and MSI logic devices, including a complete logic design of the Pep/8 ALU. Chapter 11 illustrates the fundamental concept of a finite state machine through the

state transition diagrams of sequential circuits. It concludes with a description of common computer subsystems such as bidirectional buses, memory chips, and two-port memory banks.

Level Mc2 Chapter 12 describes the microprogrammed control section of the Pep/8 CPU. It gives the control sequences for a few sample instructions and addressing modes and provides a large set of exercises for the others. It also presents concepts of load/store architectures contrasting the MIPS RISC machine with the Pep/8 CISC machine. It concludes with performance issues by describing cache memories, pipelining, dynamic branch prediction, and superscalar machines.

Use in a Course

This book offers such broad coverage that instructors may wish to omit some of the material when designing the course. Chapters 1–5 should be considered core. Selections can be made from Chapters 6 through 12.

In the book, Chapters 1–5 must be covered sequentially. Chapters 6 (Compiling to the Assembly Level) and 7 (Language Translation Principles) can be covered in either order. I often skip ahead to Chapter 7 to initiate a large software project, writing an assembler for a subset of Pep/8 assembly language, so students will have sufficient time to complete it during the semester. Chapter 11 (Sequential Circuits) is obviously dependent on Chapter 10 (Combinational Circuits), but neither depends on Chapter 9 (Storage Management), which may be omitted. Figure P.2, a chapter dependency graph, summarizes the possible chapter omissions.

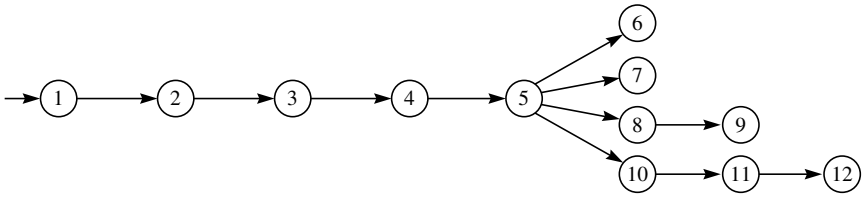


Figure **P.2**
A chapter dependency graph.

Support Materials

The support material listed below is available from the publisher's web site

`http://computersystems.jbpub.com`

Pep/8 Assembler and Simulator The Pep/8 machine is available for MS Windows, MacOS, and Unix/Linux systems. The assembler features

- an integrated text editor,
- error messages in red type that are inserted within the source code at the place where the error is detected,

- student-friendly machine language object code in hexadecimal format,
- the ability to code directly in machine language, bypassing the assembler,
- the ability to redefine the mnemonics for the unimplemented opcodes that trigger synchronous traps.

The simulator features

- simulated ROM that is not altered by load instructions,
- a small operating system burned into simulated ROM that includes a loader and a trap handler system,
- an integrated debugger that allows for break points, single step execution, CPU tracing, and memory tracing,
- the option to trace an application, the loader, or the operating system in any combination,
- a user-defined upper limit on the statement execution count to recover from endless loops,
- the ability to modify the operating system by designing new trap handlers for the unimplemented opcodes.

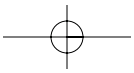
Computer Systems Figures Every figure in the book is enlarged and provided in PDF format for use in lecture presentations.

Solutions Manual Solutions to selected exercises are provided in an appendix. Solutions to the remaining exercises are available to instructors who adopt the book. For security reasons, the solutions are available only in hardcopy form directly from the publisher.

Changes to the Third Edition

This edition is a substantial revision of the previous. Many users of the second edition had requests for features to add to Pep/7, most of which were quite reasonable. Incorporating them was possible only by a complete redesign of the ISA level, which impacted virtually every chapter, many of which were entirely rewritten. There are too many improvements to list here, but the major ones are as follows:

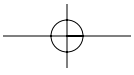
- Improved ISA level for Pep/8—The number of addressing modes has increased from four to eight, allowing simplification of the translation pattern from C++ to assembly language. The CPU is simpler, as there is no longer a base register, and translation patterns are now uniform. Some translations require fewer instructions, such as accessing global arrays and passing global variables by reference. Other translations in this edition were impossible with Pep/7, such as accessing local arrays and passing local variables by reference. The new modes permit an expanded coverage of local



and global pointers with complete programs that can allocate from a simple heap.

- Improved C++ semantics—One impact of the ISA changes is to illustrate more translation patterns. A more significant impact is that the translation process now emphasizes the C++ memory model, specifically the difference between global variables allocated at a fixed address in memory, local variables allocated on the run-time stack, and dynamic storage allocation from the heap.
- Improved assembly language syntax—The Pep/7 assembly language has a nonstandard designation for constant values. Pep/8 follows what has become the de facto standard notation for many assemblers. It adheres in general to the C/C++ and gcc conventions. Namely, decimal constants are not prefixed, hexadecimal constants are prefixed with 0x, character constants are enclosed in single quotes, string constants are enclosed in double quotes, and backslash quoting is provided in the usual way.
- Improved OS in simulated ROM—The Pep/8 operating system stored in simulated ROM has been refactored to bring two substantial improvements over the Pep/7 operating system: a new general addressing mode assertion routine and a unified operand computation routine. Both of these features not only make the operating system easier to understand, but also make it shorter because of less duplication of code. A new trap instruction provides null-terminated string output.
- Improved Pep/8 CPU design—The data section of the Pep/7 CPU has many deficiencies that are corrected in Pep/8. The biggest problem in the second edition is the computation of the status bits, which requires an inordinate number of cycles to execute. The new hardware is much more efficient and allows for more instructive examples.
- New topics—A new chapter on computer organization discusses the construction of a microprogrammed level for Pep/8. It now uses the MIPS machine to contrast load/store architectures with Pep/8-style accumulator machines, and describes cache memories, pipelining, superscalar machines, and many other hardware topics not in the second edition.
- Improved historical coverage—New sidebars in this edition cover influential people in the history of computing. Each chapter highlights one or two people who were instrumental in developing some aspect of computing described in that chapter. The historical narrative is woven around the biographical information of the people.

A unique feature of the book that is retained in the third edition is its breadth of topics. With the additional depth of coverage in the compiler translation process, it is of greater use to those who continually ask the question, “What is the place of assembly language programming in the computer science curriculum?” The third edition gives the same answer as the second—to provide a depth of understanding about



the architecture of the ubiquitous von Neumann machine. The additional chapter on computer organization makes the new edition a better contender for the computer organization course as well. However, *Computer Systems* retains its unique goal to provide a balanced overview of all the main areas of the field, including the integration of software and hardware and the integration of theory and practice.

Computing Curricula 2001

The ACM and IEEE Computer Society have established Curriculum 2001 guidelines for Computer Science. The guidelines present a taxonomy of bodies of knowledge with a specified core. *Computer Systems* applies to the category Architecture and Organization (AR) and covers practically all of the core topics from the AR body of knowledge. The AR core areas from the preliminary report, together with the chapters from this text that cover each area, are

AR1. Digital logic and digital systems, Chapters 10, 11, 12

AR2. Machine level representation of data, Chapter 3

AR3. Assembly level machine organization, Chapters 4, 5, 6

AR4. Memory system organization and architecture, Chapters 9, 11

AR5. Interfacing and communication, Chapters 8, 9

AR6. Functional organization, Chapters 11, 12

AR7. Multiprocessing and alternative architectures, Chapter 8

Acknowledgments

Pep/1 had 16 instructions, one accumulator, and one addressing mode. Pep/2 added indexed addressing. John Vannoy wrote both simulators in ALGOL W. Pep/3 had 32 instructions and was written in Pascal as a student software project by Steve Dimse, Russ Hughes, Kazuo Ishikawa, Nancy Brunet, and Yvonne Smith. In an early review, Harold Stone suggested many improvements to the Pep/3 architecture that were incorporated into Pep/4 and carried into later machines. Pep/4 had special stack instructions, simulated ROM, and software traps. Pep/5 was a more orthogonal design, allowing any instruction to use any addressing mode. John Rooker wrote the Pep/4 system and an early version of Pep/5. Gerry St. Romain implemented a Mac-OS version and an MS-DOS version. Pep/6 simplified indexed addressing and included the complete set of conditional branch instructions. John Webb wrote the trace facility using the BlackBox development system. Pep/7 increased the installed memory from 4 MBytes to 32 MBytes. Pep/8 increased the number of addressing modes from four to eight, and the installed memory to 64 KBytes. The GUI version of the Pep/8 assembler and simulator was implemented in C++ by a team of students

using the Qt development system. The team, led by Ryan Okelberry, included Deacon Bradley, Jeff Cook, Nathan Counts, Stuartt Fox, Hermi Heimgartner, Thomas Rampelberg, Di Wang, and Matt Wells. Matt Highfield rewrote the graphical user interface. Ryan Okelberry also wrote the Pep/8 CPU simulator. Luciano d'Ilori wrote the command line version of the assembler.

More than any other book, Tanenbaum's *Structured Computer Organization* has influenced this text. This text extends the level structure of Tanenbaum's book by adding the high-order programming level and the applications level at the top.

The following reviewers of the manuscript and users of the previous edition shaped the final product significantly: Wayne P. Bailey, Jim Bilitski, Fadi Deek, William Decker, Peter Drexel, Gerald S. Eisman, Victoria Evans, David Garnick, Ephraim P. Glinert, Dave Hanscom, Michael Hennessy, Michael Johnson, Andrew Malton, Robert Martin, Richard H. Mercer, Randy Molmen, John Motil, Peter Ng, Bernard Nudel, Carolyn Oberlink, Wolfgang Pelz, James F. Peters III, James C. Pleasant, Eleanor Quinlan, Glenn A. Richard, David Rosser, Gerry St. Romain, Harold S. Stone, J. Peter Weston, and Norman E. Wright. Joe Piasentin provided artistic consultation. Two people who influenced the design of Pep/8 significantly are Myers Foreman, who was a source of many ideas for the instruction set, and Douglas Harmes, who suggested among other improvements the `MOVSPA` instruction that makes possible the passing of local variables by reference.

At Jones and Bartlett Publishers, Editor Stephen Solomon, Production Editor Amy Rose, and Editorial Assistant Deborah Arrand provided valuable support and were a true joy to work with. Kristin Ohlin captured the flavor of the book with her striking cover design.

I am fortunate to be at an institution that is committed to excellence in undergraduate education. Pepperdine University, in the person of Ken Perrin, provided the creative environment and the professional support in which the idea behind this project was able to evolve. My wife, Ann, provided endless personal support. To her I owe an apology for the time this project has taken, and my greatest thanks.

Stan Warford
Malibu, California